UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/648,791	08/27/2003	Gerard Damm	3448-Z	4488

7590 06/23/2006

Law Office of Jim Zegeer
Suite 108
801 North Pitt Street
Alexandria, VA 22314

EXAMINER	
WONG, JOSEPH D	

ART UNIT	PAPER NUMBER
2195	

DATE MAILED: 06/23/2006

Please find below and/or attached an Office communication concerning this application or proceeding.

UNITED STATES DEPARTMENT OF COMMERCE
U.S. Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450

10/678,791

APPLICATION NO./ CONTROL NO.	FILING DATE	FIRST NAMED INVENTOR / PATENT IN REEXAMINATION	ATTORNEY DOCKET NO.

EXAMINER	
ART UNIT	PAPER
	20060615

DATE MAILED:

Please find below and/or attached an Office communication concerning this application or proceeding.

Commissioner for Patents

June 15, 2006

Jim Zegeer, Esq.
Law Office of Jim Zeeger
801 North Pitt Street #108
Alexandira, VA 22314

Dear Mr. Zegeer:

Your legal secretary Pat informed us that referenced pages were missing from the mailing you received. If you received two courtesy faxes and one e-mail from my assistant dated June 15, these are duplicates being mailed as a formality.

One piece of uncited art my assistant noticed today, Gaede, Volker et al., "Multidimensional access methods", June 1998, ACM Computing Surveys (CSUR), P. 17-172, 182, 216, 231. This is included as a summary of tree structure development prior to June 1998.

Sincerely,

JW

DAVID V. BRUCE
PRIMARY EXAMINER

On-Demand Submaps

One of the potential problems with having maps that contain thousands of nodes involves the use of available memory.

The `ipmap` application[5] typically creates a hierarchy of several levels of submaps (root, Internet, segment, node). An interesting and important point is that many of these submaps are rarely, if ever, visited by an operator.

Imagine if NNM kept every submap in the map in memory at all times. In a large map, with literally thousands of submaps, this approach would create a substantial requirement for memory.

However, NNM makes it possible to specify that only some of the submaps be kept persistently in memory; any other submap is created if and only if a user requests it. This means that for a management station with a given amount of memory, you can manage maps with many more submaps containing many more objects.

The main benefit of the on-demand submap feature is a dramatic improvement in performance, especially during synchronization. Meanwhile, the features and functionality of NNM are essentially identical whether or not on-demand submaps are enabled. Whenever you double-click on an explodable object in `ipmap`, the submap opens promptly (though submaps with many objects may take several seconds to open). Furthermore, all features of NNM, including "find" and IP status propagation, work as usual.

Some applications that were created before this feature was part of NNM may not be compatible with the on-demand submap feature. They may expect certain objects to be available in memory at all times. In this case, you can use persistence filtering to make those objects persistent.[6] Persistence filtering provides per-object exceptions to the on-demand feature, so that certain objects and their submaps are always present in memory even if on-demand submaps are enabled.

5. See Managing Your Network with HP OpenView Network Node Manager for information about this and other key NNM processes.
6. See "Persistence Filtering" on page 71 for details.

Distributed Threshold Monitoring and Event Forwarding

In a distributed system, it is necessary that operators at the management console be aware of critical events wherever they occur. This is commonly called "management by exception." For this reason, NNM has the ability to forward events, typically from collection stations to management stations.

Using the graphical `Event Configuration` window, you can configure any event, on a case-by-case basis, to be forwarded to any or all of the following:

- All managers currently using the NNM station as a collection station.
- Particular hosts, identified by hostname or IP address.
- Hosts listed (by hostname or IP address) in a specific file.

All configurable events, including threshold events, can be configured for forwarding. Internal NNM events are automatically forwarded to management stations as needed.

When defining new events, we recommend that you use multiple varbinds with individual pieces of data rather than using a full string. This allows the receiving machine to specify the formatting. Refer to the reference page in NNM online help (or the UNIX system manpage) for *trapd.conf* more information.

General Events

You can forward any or all events from collection stations to management stations. This is most useful when a collection station receives little or no local attention, or when you have applications that issue events which you want to have forwarded to the management station.

For example, suppose you have a collection station that monitors an agent that generates an event called `FuseIsLit`. At the collection station, you can configure that event to be forwarded to the management station, and thereby alert the operator that the device is not functioning properly.

Threshold Events

Threshold events give you a way to be notified of traffic patterns that are outside normal expectations. For example, you could set a threshold to monitor disk utilization on a server. If the amount of free disk falls below the threshold you set,

NNM can notify an operator, or take other predetermined actions (such as mailing a message). Threshold events help operators detect and isolate problems as they arise, before users experience difficulty.

In the context of a distributed network management solution, the ability to forward threshold events means that the management station does not itself have to perform data-collection on all the objects that need thresholds monitored. That effort, and the associated network traffic, is delegated to the various collection stations that the manager uses.

Trend Data Collection in a Distributed NNM Solution

Some users may be interested in having trend data collected at remote locations and forwarded back to a central point, such as an NNM management station, for trend analysis.

Distributed data collection doesn't reduce the load on a management station if the data is sent back to the management station in real time. Therefore, in NNM the remote station stores any trend data in its own database. Data can then be transferred to a relational database if that is available.[7]

NNM does not provide a way to automatically synchronize data back to the management station. It does, however, offer some basic tools that you can use to create custom solutions. For example, you can transfer NNM's data files from the collection stations to the management station (via FTP or a similar utility) during off-peak periods. Then you can use NNM's data-merging tool (see the *ovcoltosql* reference page in NNM online help (or the UNIX system manpage)) to merge the data into an SQL database. Then you can use database tools to create reports on the data.

7. See Reporting and Data Analysis with HP OpenView Network Node Manager for information on supported database.

Large-Map Viewing Support

When a submap contains hundreds of nodes, the symbols and connections get smaller, and it becomes difficult to make sense of the graphical picture. Also, changes in status are not as noticeable.

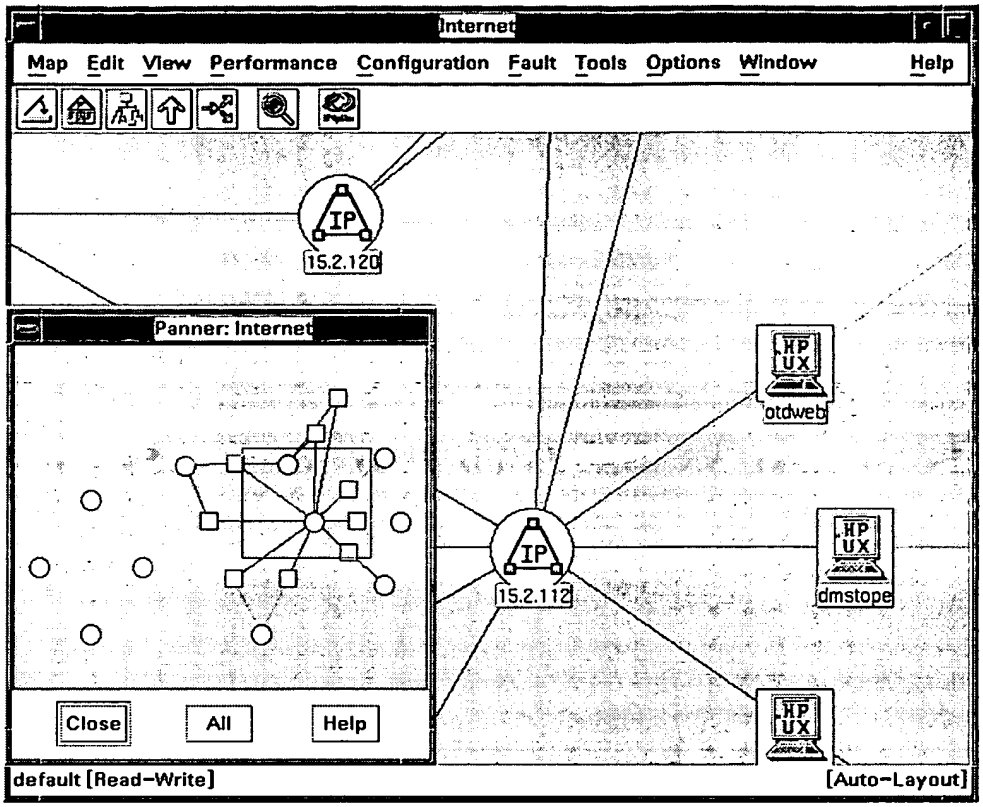
HP OpenView Network Node Manager has two features that make it easy to use maps that contain hundreds, even thousands, of nodes:

- The Panner
- The Quick Navigator

The Panner

The panner, shown in Figure 1-3, provides a way to zoom in on regions of a submap, and drag the zoomed region around. See Managing Your Network with HP OpenView Network Node Manager, or NNM's online help, for details.

Figure 1-3 **The Panner**

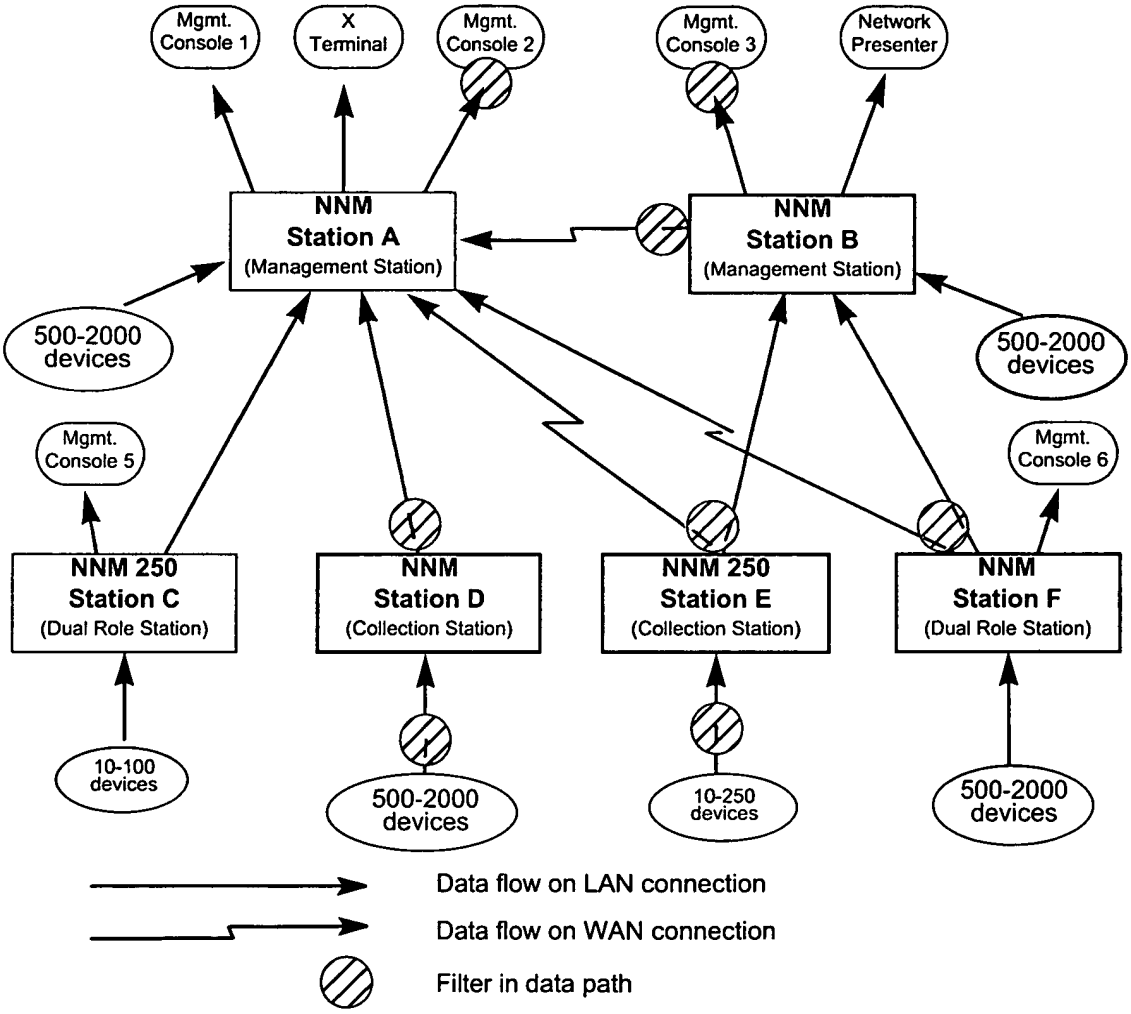


The Quick Navigator

The Quick Navigator is a special window that you can customize easily for quick access to your most frequently visited submaps. Your tool bar has a Quick Navigator button, which is described in the online help.

Each operator can customize their own Quick Navigator window by following the simple directions in the online help.

Figure 2-1 The Distribution Model for Network Node Manager



With NNM, it is also possible to obtain management information from one or more NNM stations acting as collection stations. In Figure 2-1, most of the NNM stations (Stations B, C, D, E, and F) are operating in the role of collection stations.

Stations C and F illustrate how an NNM station can have the role of a management station (possibly making use of collection stations), a collection station (serving one or more management stations), or both roles at once.[3]

3. See "About Management Stations and Collection Stations" on page 27.

It is possible for a collection station to serve more than one management station; this is illustrated by NNM 250 Station E in Figure 2-1, and also by NNM Station F.

Network Connections between Distributed Components

The connection between a management station and a collection station can be either WAN or LAN-based. WAN connections to collection stations are common. When part of the network you need to manage is remote (that is, accessible via WAN rather than LAN connections), you should consider setting up a collection station at the remote site; distributing NNM will minimize traffic on the WAN connection while providing full network management functionality.

In contrast, it is important to note that a management console must have a LAN connection to its management station. This is because several NNM components (including the event system, topology manager, and HP OpenView Windows database) require high speed communication between the management station and the management console. This performance has been achieved on UNIX systems by using NFS to implement management consoles, and this is the reason management consoles are not supported across WAN links. NNM on Windows NT operating system uses Windows NT operating system file sharing for remote access to the files (for the case of a Windows NT operating system console to a Windows NT operating system management station), and, while WAN links are supported, they are slow.

In Figure 2-1, the LAN and WAN links between management and collection stations were arbitrarily chosen for illustrative purposes. In contrast, the links from management consoles to management stations are necessarily LAN connections.

Filters

The function of most filters is to select network management data that is of sufficient value to be kept, and to eliminate other data.[4] Data that is useful at one point (say, a site management station) can be filtered out before it gets sent to another (say, regional management station) where it is unneeded.

NNM uses two basic types of filters:

- A data-stream filter acts as a sieve through which objects flow. This type of filter contains criteria that permit some objects to proceed, and blocks other objects. Downstream from a data-stream filter, the data stream includes only those objects that the filter passed; other objects are blocked, and thus eliminated from the data at the point of the filter.

  Note that once an object has passed a data-stream filter, later changes to the filter have no effect on it. It remains in the data stream.

- A set-defining filter is applied to a pool of candidate objects, and determines which objects belong in the final set based on the criteria contained in the filter. Before a new object can be added to the set, it is tested by the set-defining filter to determine if it is a valid member or not.

  If a set-defining filter is changed, all candidate objects are reevaluated to determine which ones belong in the new set and which do not. Then the set is reconstituted out of valid members.

The difference may seem abstract at first. As an analogy, you can compare a data-stream filter to a wire-mesh sieve, of the kind used to sort stones. Once a large stone has passed through the sieve, it is in the sorted pile beneath regardless of whether you later switch to a finer mesh in the sieve.

Imagine, on the other hand, that your sieve behaved like a set-defining filter. In that case, changing to a finer mesh in the sieve would cause all the source material to be refiltered, yielding a sorted pile of uniformly smaller stones.

In either case, remember that a filter specifies or describes the objects you want it to pass, not the objects you want it to reject. When the filter is applied, it passes the selected objects through, eliminating the rest from the data stream. As a user, you can apply filters only to object data; however, applications can filter events also.[5]

4. See "Persistence Filtering" on page 71 for an important exception.
5. Interested application developers should see the *HP OpenView Integration Series: SNMP Developer's Guide.*

Filters

Filtering is based on the information contained in a subset of the fields of the object database (ovwdb).[6] These fields are candidates to be all or part of the criteria by which objects are assessed in a filter.

Filters can eliminate unnecessary data at different points in the system:

- Discovery filtering at any NNM station.
- Exported topology filtering at the collection station.
- Map filtering at the management station.
- Failover filtering on the management station.

Table 2-1 compares the types of filters. See "Filter File Example" on page 173 for more information.

6. See "Filterable Objects and Attributes" on page 166 for specifics.

Table 2-1 Comparing Types of Filters

Filter Type	Apply on Incoming Data	Reevaluate If Filter Changes	Purpose	Applied So Data Is	Reference
Discovery	X		Limit scope of objects added to database.	Excluded	page 46
Topology	X	X	Limit information forwarded from collection station to management station.	Excluded	page 47
Map	X	X	Show only items of interest to operator on map.	Excluded	page 48
Persistence	X	X	Disable on-demand submaps for third party applications.	Included	page 71
Failover	X	X	Limit nodes polled by management station when collection station fails.	Included	page 56
Important Nodes	X	X	List unreachable important nodes as primary alarms in the Alarm Browser rather than children of a connector down event.	Included	Managing Your Network with HP OpenView Network Node Manager
DHCP Range	X		Identify DHCP (shared or floating) IP addresses	Included	Managing Your Network with HP OpenView Network Node Manager

Discovery Filtering

Discovery filters specify which devices an NNM station is actively discovering and monitoring. The purpose of discovery filtering is to reduce the discovery and monitoring effort of the station. Different stations have independent (though possibly identical) discovery filters.

Discovery filtering limits the scope of objects that `netmon` adds to the collection station topology database. To unmanage objects and limit the set of nodes that are polled at all, refer to Managing Your Network with HP OpenView Network Node Manager. The filter may be set to pass, for example:

- Gateways.
- Bridges, routers, and hubs.
- All devices.
- Nodes based on their `sysObjectID`.
- Objects inside or outside of a particular range of IP addresses.

By default, Segments and Networks pass the discovery filter.

Discovery filtering is achieved by configuring the `netmon` service[7]; the filter is then applied to all newly discovered objects. Objects that are rejected by the discovery filter never appear in any topology or object database.

Discovery filters are data-stream filters; changes to a discovery filter affect new data *only*. All objects that previously passed the filter remain in the data stream, regardless of whether they would currently pass or not, and polling is still done on all previously discovered objects whether or not they would now pass the filter. You can, however, use the `ovtopofix` command to change the set of previously discovered objects.[8]

Implement any discovery filtering on an NNM station before you begin using it as a collection station; this will improve overall performance, by reducing the amount of synchronization effort.

7. See "Configuring a Discovery Filter" on page 123, for details, *netmon* and *xnmpolling* reference pages in NNM online help (or the manpages on UNIX systems), the NNM online help, and "Distributed Services Architecture" on page 52.
8. See the *ovtopofix* reference page in NNM online help (or the UNIX system manpage) for details.

Topology Filtering

Topology filters specify which topology data gets forwarded to a management station. The result is less management traffic on the network and lower data storage requirements at the management station. By default, the topology filter does not pass Networks and Segments.

A topology filter at a collection station defines the subset of topology data that management stations can see. The idea is to have the topology filter at the collection station pass information about only those objects in which the manager is interested. Data about objects outside the manager's interest doesn't clog up the link to the management station.

Objects rejected by a topology filter remain in the collection station's topology database. They are not, however, provided to a higher-level management station. Each collection station has only one topology filter in place. Note that this means the topology filter of a collection station must meet the needs of all management stations.Through topology filtering, you can choose to provide the following kinds of topology data to interested managers (many other choices exist):[9]

- All topology data, by using no filter at all.
- Nodes with (or specifically without) a certain `sysObjectID`.
- Objects inside or outside of a particular range of IP addresses.
- Nodes from a particular vendor.
- Objects with particular physical address.
- All token ring segments.

Topology filtering is achieved by configuring the `ovtopmd` service.[10] You should test topology filters at the collection station before the station goes into service.[11]

Topology filters are, technically, data-stream filters. However, in effect, any change to a topology filter affects new and old data alike. When a topology filter is changed, the topology database of the management station is resynchronized with the filtered collection station topology database. Some objects may be deleted from the

9. See "Filterable Objects and Attributes" on page 166 for specifics.
10. See "Distributed Internet Discovery and Monitoring" on page 51, Chapter 4, "Procedures for Scaling NNM," and the *ovtopmd* reference page in NNM online help (or the UNIX system manpage), for further details.
11. See the reference page in NNM online help (or the UNIX system manpage) for *ovtopodump*, especially the `-f` option, for instructions on testing topology filters.

management station topology (if they do not pass the new topology filter), and some objects may be added (if they were previously rejected, but now pass the topology filter). As a result, the database of the management station ends up reflecting the topology being passed by the new topology filter, so the effect is that of a set-defining filter.

NOTE

Topology filters do not affect the events that are passed from collection stations to management stations. This means that any events that you have configured to be forwarded are in fact forwarded from the collection station to the management stations, regardless of whether the objects to which the events refer pass the topology filter or not. Listing particular event sources in the event configuration at the management station can reduce the likelihood of operator confusion.

Map Filtering

The purpose of map filtering is to give the operator visibility of only those objects in which he or she has interest. By default, Networks and Segments do not pass the map filter.

Map filtering occurs on a per-map basis, not a per-display basis. All operators who use a particular map have the same filtered view. Additional maps can be created and separately filtered for specific purposes.

Objects that are rejected by a map filter remain in the management station's topology database, and are still subject to status polling. Events from these objects pass through and are visible in the event subsystem, but this can be changed to show only events from objects on the operator's current map.[12]

Like the previously discussed types of filtering, map filtering can be configured so that (regardless of how much other topology data exists in the database) the map seen by the operator displays only the objects of interest to that operator. In general, this means objects that match some combination of attributes and attribute values; among the many choices, this can include:[13]

- All objects, by using no filter at all.
- Connectors, networks, and segments only.
- Nodes with specific `sysObjectIDs`.
- Bridges, hubs, and their segments.
- Objects inside or outside of a particular range of IP addresses.
- Only nodes that support SNMP.

12.See the *xnmevents* reference page for details on the "filter by map" feature.
13. See "Filterable Objects and Attributes" on page 166 for specifics.

Knuth (3rd ed)

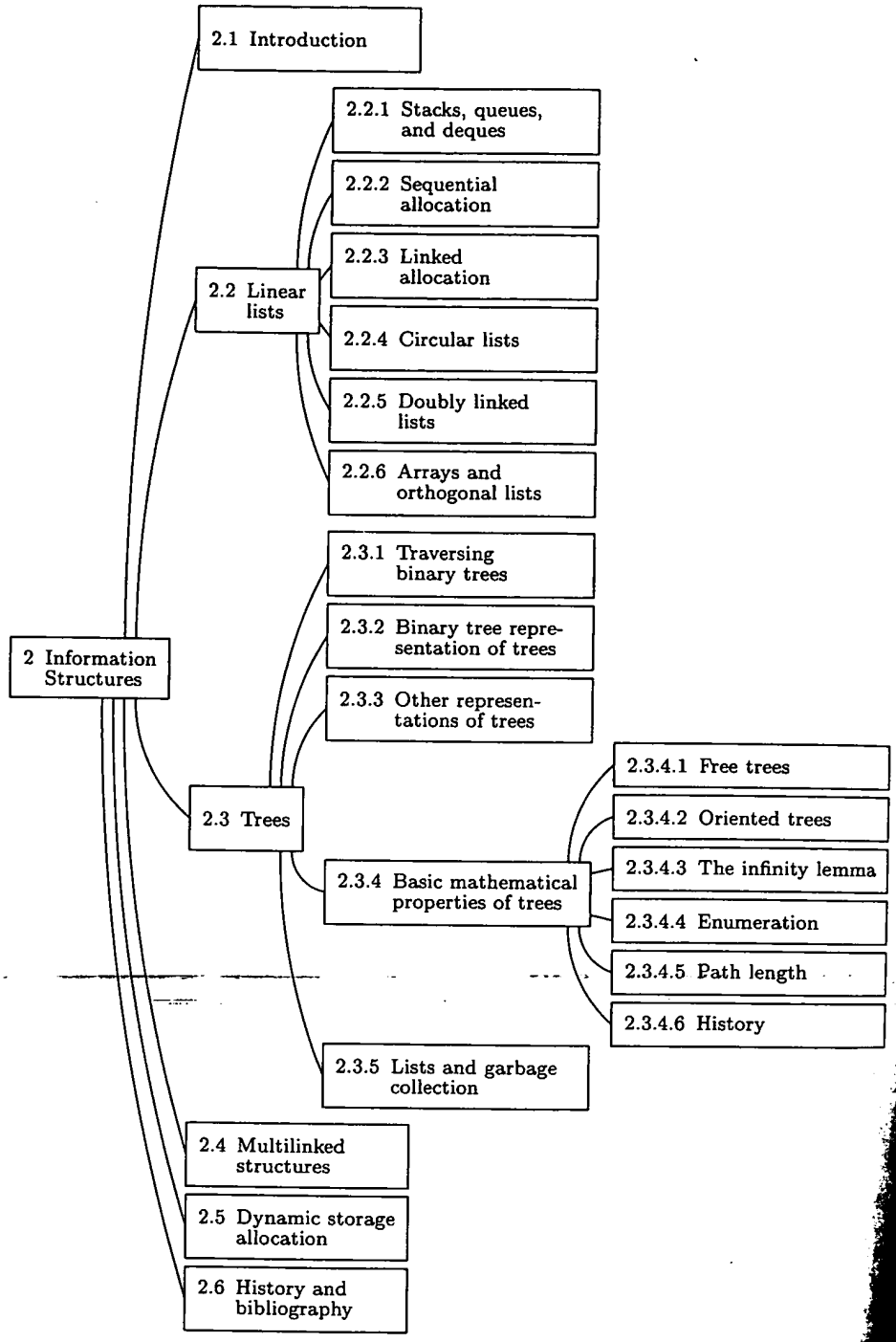
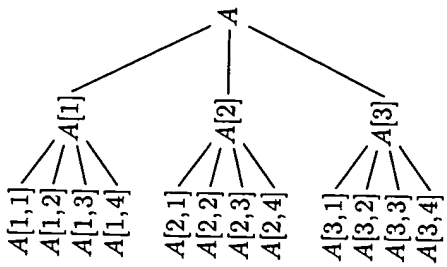


Fig. 22. The structure of Chapter 2.

Thus, in particular, we see that any rectangular array can be thought of as a special case of a tree or forest structure. For example, here are two representations of a 3×4 matrix:

$$\begin{pmatrix} A[1,1] & A[1,2] & A[1,3] & A[1,4] \\ A[2,1] & A[2,2] & A[2,3] & A[2,4] \\ A[3,1] & A[3,2] & A[3,3] & A[3,4] \end{pmatrix}$$



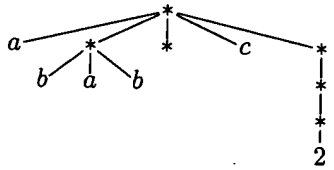
It is important to observe, however, that this tree structure does not faithfully reflect all of the matrix structure; the row relationships appear explicitly in the tree but the column relationships do not.

A forest can, in turn, be regarded as a special case of what is commonly called a *list structure*. The word "list" is being used here in a very technical sense, and to distinguish the technical use of the word we will always capitalize it: "List." A List is defined (recursively) as *a finite sequence of zero or more atoms or Lists*. Here "atom" is an undefined concept referring to elements from any universe of objects that might be desired, so long as it is possible to distinguish an atom from a List. By means of an obvious notational convention involving commas and parentheses, we can distinguish between atoms and Lists and we can conveniently display the ordering within a List. As an example, consider

$$L = (a,\ (b, a, b),\ (\,),\ c,\ (((2)))), \tag{3}$$

which is a List with five elements: first the atom a, then the List (b, a, b), then the empty List $()$, then the atom c, and finally the List $(((2)))$. The latter List consists of the List $((2))$, which consists of the List (2), which consists of the atom 2.

The following tree structure corresponds to L:



(4)

The asterisks in this diagram indicate the definition and appearance of a List, as opposed to the appearance of an atom. Index notation applies to Lists as it does to forests; for example, $L[2] = (b, a, b)$, and $L[2, 2] = a$.

No data is carried in the nodes for the Lists in (4) other than the fact that they are Lists. But it is possible to label the nonatomic elements of Lists with information, as we have done for trees and other structures; thus

$$A = (a{:}(b, c),\ d{:}(\,))$$

transpose[1] *n.* The result of rotating a matrix.

transpose[2] *vb.* **1.** To reverse, as the order of the letters *h* and *t* in *hte,* in correcting the spelling of *the;* or reversing two wires in a circuit. **2.** In mathematics and spreadsheets, to rotate a matrix (a rectangular array of numbers) about a diagonal axis.

transputer \trans`pyd´tər\ *n.* Short for **transistor computer.** A complete computer on a single chip, including RAM and an FPU, designed as a building block for parallel computing systems.

trap[1] *n. See* interrupt.

trap[2] *vb.* To intercept an action or event before it occurs, usually in order to do something else. Trapping is commonly used by debuggers to allow interruption of program execution at a given spot. *See also* interrupt, interrupt handler.

trapdoor *n. See* back door.

trap handler *n. See* interrupt handler.

Trash *n.* An icon on the screen in the Macintosh Finder, resembling a garbage can. To delete a file or eject a diskette, the user drags the icon for the file or diskette to the Trash. However, until the user shuts down the system or chooses the menu option "Empty Trash," a file in the Trash is not actually deleted; the user can retrieve it by double-clicking the Trash icon and dragging the file's icon out of the resulting window. *Compare* Recycle Bin.

traverse *vb.* In programming, to access in a particular order all of the nodes of a tree or similar data structure.

tree *n.* A data structure containing zero or more nodes that are linked together in a hierarchical fashion. If there are any nodes, one node is the root; each node except the root is the child of one and only one other node; and each node has zero or more nodes as children. *See also* child (definition 2), graph, leaf, node (definition 3), parent/child (definition 2), root.

tree network *n.* A topology for a local area network (LAN) in which one machine is connected to one or more other machines, each of which is connected to one or more others, and so on, so that the structure formed by the network resembles that of a tree. See the illustration. *See also* bus network, distributed network, ring network, star network, token ring network, topology.

tree search *n.* A search procedure performed on a tree data structure. At each step of the search, a tree search is able to determine, by the value in a particular node, which branches of the tree to eliminate, without searching those branches themselves. *See also* branch (definition 1), tree structure.

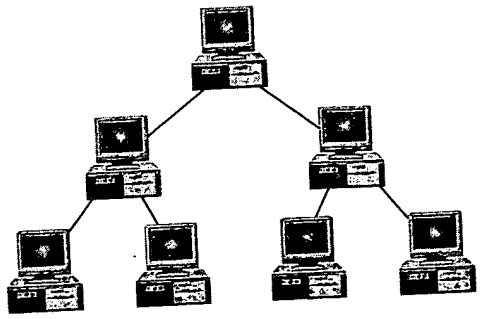
tree structure *n.* Any structure that has the essential organizational properties of a tree. *See also* tree.

trellis-coded modulation *n.* An enhanced form of quadrature amplitude modulation that is used by modems that operate at or above 9,600 bps (bits per second). Trellis-coded modulation encodes information as unique sets of bits associated with changes in both the phase and amplitude of the carrier, as well as uses extra signal points for error-checking bits. *Acronym:* TCM. *See also* quadrature amplitude modulation.

triage[1] \tre-azh^\ *n.* The process of prioritizing projects or elements of a project (such as bug fixes) to ensure that available resources are assigned in the most effective, time-efficient, and cost-efficient manner. Traditionally, triage has referred to the prioritization of treatment to the wounded during wartime or medical disaster situations. More recently, the term also refers to anticipating and preventing computer system crashes brought on by the Year 2000 (Y2K) problem. *See also* Year 2000 Problem.

triage[2] \tre-azh^\ *vb.* To identify and prioritize the elements of a project or problem to order them in a way that makes best use of labor, funds, and other resources.

trichromatic \tri`krə-mat´ik\ *adj.* Of, pertaining to, or characteristic of a system that uses three colors (red, green, and blue in computer graphics) to create all other colors. *See also* color model.



Tree network.

Multidimensional Access Methods

VOLKER GAEDE

IC-Parc, Imperial College, London

AND

OLIVER GÜNTHER

Humboldt-Universität, Berlin

Search operations in databases require special support at the physical level. This is true for conventional databases as well as spatial databases, where typical search operations include the *point query* (find all objects that contain a given search point) and the *region query* (find all objects that overlap a given search region). More than ten years of spatial database research have resulted in a great variety of multidimensional access methods to support such operations. We give an overview of that work. After a brief survey of spatial data management in general, we first present the class of *point access methods*, which are used to search sets of points in two or more dimensions. The second part of the paper is devoted to *spatial access methods* to handle extended objects, such as rectangles or polyhedra. We conclude with a discussion of theoretical and experimental results concerning the relative performance of various approaches.

Categories and Subject Descriptors: H.2.2 [**Database Management**]: Physical Design—*access methods*; H.2.4 [**Database Management**]: Systems; H.2.8 [**Database Management**]: Database Applications—*spatial databases and GIS*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*search process, selection process*

General Terms: Design, Experimentation, Performance

Additional Key Words and Phrases: Data structures, multidimensional access methods

1. INTRODUCTION

With an increasing number of computer applications that rely heavily on multidimensional data, the database community has recently devoted considerable attention to spatial data management. Although the main motivation originated in the geosciences and mechanical CAD, the range of possible applications has expanded to areas such as robotics, visual perception, autonomous navigation, environmental protection, and medical imaging [Günther and Buchmann 1990].

The range of interpretation given to

This work was partially supported by the German Research Society (DFG/SFB 373) and by the ESPRIT Working Group CONTESSA (8666).

Authors' address: Institut für Wirtschaftsinformatik, Humboldt-Universität zu Berlin, Spandauer Str. 1, 10178 Berlin, Germany; email: ⟨{gaede,guenther}@wiwi.hu-berlin.de⟩.

CONTENTS

the term *spatial data management* is just as broad as the range of applications. In VLSI CAD and cartography, this term refers to applications that rely mostly on two-dimensional or layered two-dimensional data. VLSI data are usually represented by rectilinear polylines or polygons whose edges are isooriented, that is, parallel to the coordinate axes. Typical operations include intersection and geometric routing [Shekhar and Liu 1995]. Cartographic data are also two-dimensional with points, lines, and regions as basic primitives. In contrast to VLSI CAD, however, the shapes are often characterized by extreme irregularities. Common operations include spatial searches and map overlay, as well as distance-related operations. In mechanical CAD, on the other hand, data objects are usually three-dimensional solids. They may be represented in a variety of data formats, including cell decomposition schemes, constructive solid geometry (CSG), and boundary representations [Kemper and Wallrath 1987]. Yet other applications emphasize the processing of unanalyzed images, such as X-rays and satellite imagery, from which features are extracted. In those areas, the terms *spatial database* and *image database* are sometimes even used interchangeably.

Strictly speaking, however, spatial databases contain multidimensional data with explicit knowledge about objects, their extent, and their position in space. The objects are usually represented in some vector-based format, and their relative positions may be explicit or implicit (i.e., derivable from the internal representation of their absolute positions). Image databases often place less emphasis on data analysis. They provide storage and retrieval for unanalyzed pictorial data, which are typically represented in some raster format. Techniques developed for the storage and manipulation of image data can be applied to other media as well, such as infrared sensor signals or sound.

In this survey we assume that the goal is to manipulate analyzed multidimensional data and that unanalyzed images are handled only as the source from which spatial data can be derived. The challenge for the developers of a spatial database system lies not so much in providing yet another collection of special-purpose data structures. Rather, one has to find abstractions and architectures to implement generic systems, that is, to build systems with generic spatial data-management capabilities that can be tailored to the requirements of a particular application domain. Important issues in this context include the handling of spatial representations and data models, multidimensional access methods, and pictorial or spatial query languages and their optimization.

This article is a survey of multidimensional access methods to support search operations in spatial databases. Figure 1, which was inspired by a similar graph by Lu and Ooi [1993], gives a first overview of the diversity of existing multidimensional access methods. The goal is not to describe all of these structures, but to discuss the most prominent ones, to present possible taxonomies, and to establish references to other literature.

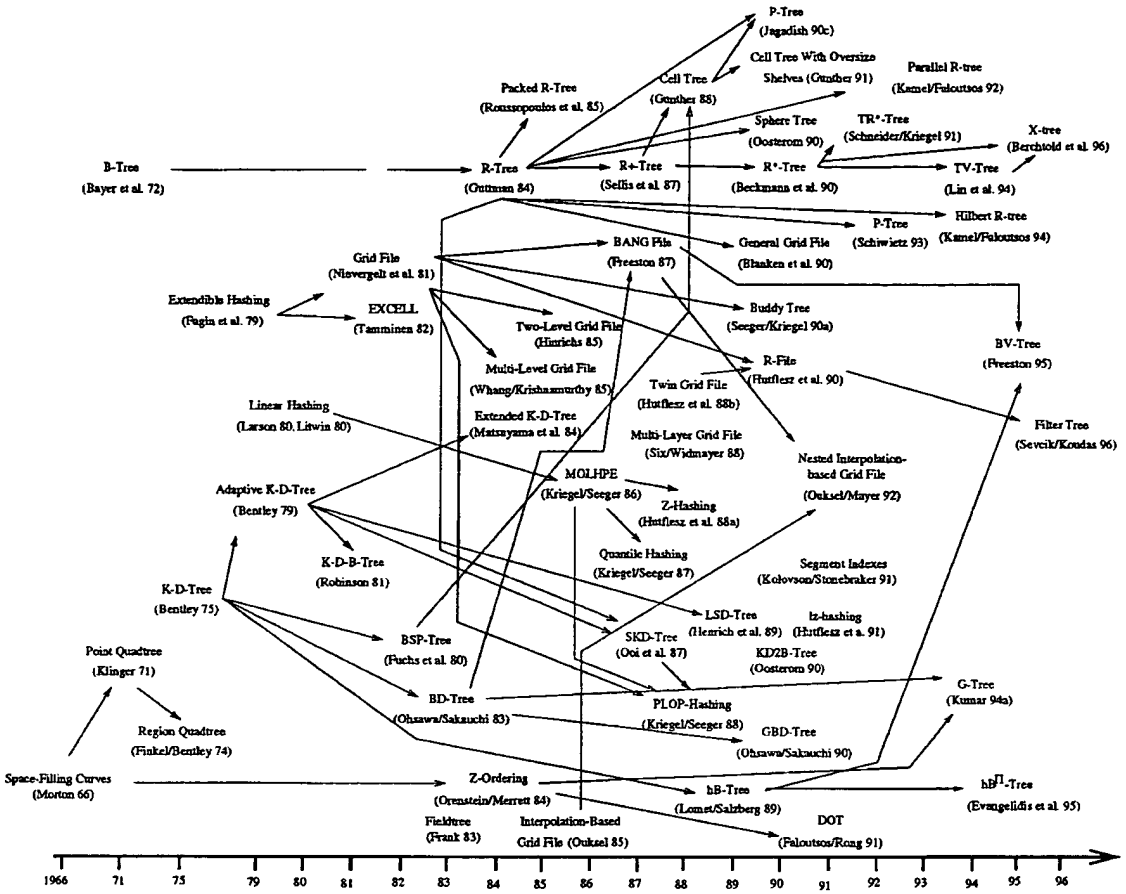


Figure 1. History of multidimensional access methods.

Several shorter surveys have been published previously in various Ph.D. theses such as Ooi [1990], Kolovson [1990], Oosterom [1990], and Schiwietz [1993]. Widmayer [1991] gives an overview of work published before 1991. Like the thesis by Schiwietz, however, his survey is available only in German. Samet's books [1989, 1990] present the state of the art until 1989. However, they primarily cover quadtrees and related data structures. Lomet [1991] discusses the field from a systems-oriented point of view.

The remainder of the article is organized as follows. Section 2 discusses some basic properties of spatial data and their implications for the design and implementation of spatial databases. Section 3 gives an overview of some traditional data structures that had an impact on the design of multidimensional access methods. Sections 4 and 5 form the core of this survey, presenting a variety of point access methods (PAMs) and spatial access methods (SAMs), respectively. Some remarks about theoretical and experimental analyses are contained in Section 6, and Section 7 concludes the article.

2. ORGANIZATION OF SPATIAL DATA

2.1 What Is Special About Spatial?

To obtain a better understanding of the requirements in spatial database systems, we first discuss some basic properties of spatial data. First, spatial data have a complex structure. A spatial data object may be composed of a single point or several thousands of polygons,

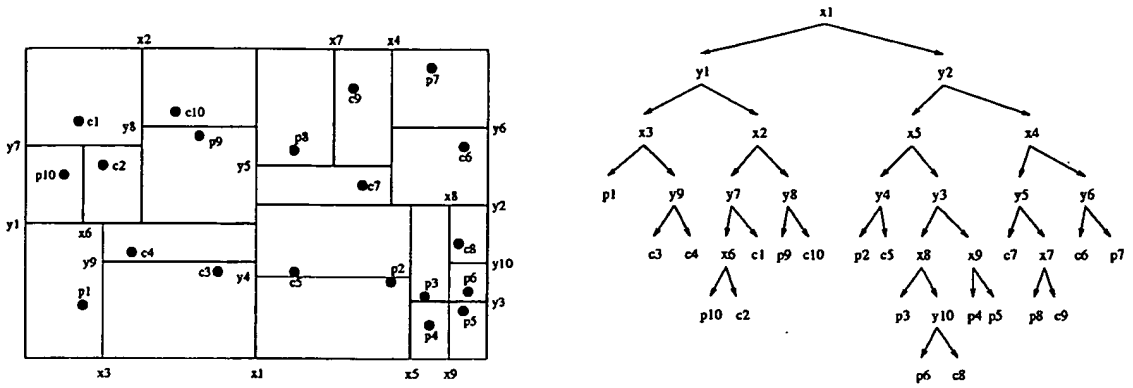


Figure 11. Adaptive k-d-tree.

priori and if updates are rare. Figure 11 shows an adaptive k-d-tree for the running example. Note that the tree still depends on the order of insertion.

Another variant of the k-d-tree is the *bintree* [Tamminen 1984]. This structure partitions the universe recursively into d-dimensional boxes of equal size until each one contains only a certain number of points. Even though this kind of partitioning is less adaptive, it has several advantages, such as the implicit knowledge of the partitioning hyperplanes. In the remainder of this article, we encounter several other structures based on this kind of partitioning.

A disadvantage common to all k-d-trees is that for certain distributions no hyperplane can be found that splits the data points evenly [Lomet and Salzberg 1989]. By introducing a more flexible partitioning scheme, the following BSP-tree avoids this problem completely.

3.2.2 *The BSP-Tree [Fuchs et al. 1980, 1983].* Splitting the universe only along iso-oriented hyperplanes is a severe restriction in the schemes presented so far. Allowing arbitrary orientations gives more flexibility to find a hyperplane that is well suited for the split. A well-known example for such a method is the *binary space partitioning (BSP)-tree*. Like k-d-trees, BSP-trees are binary trees that represent a recursive subdivision of the universe into subspaces by means of $(d - 1)$-dimensional hyperplanes. Each subspace is subdivided independently of its history and of the other subspaces. The choice of the partitioning hyperplanes depends on the distribution of the data objects in a given subspace. The decomposition usually continues until the number of objects in each subspace is below a given threshold.

The resulting partition of the universe can be represented by a BSP-tree in which each hyperplane corresponds to an interior node of the tree and each subspace corresponds to a leaf. Each leaf stores references to those objects that are contained in the corresponding subspace. Figure 12 shows a BSP-tree for the running example with no more than two objects per subspace.

In order to perform a point query, we insert the search point into the root of the tree and determine on which side of the corresponding hyperplane it is located. Next, we insert the point into the corresponding subtree and proceed recursively until we reach a leaf of the tree. Finally, we examine the data objects in the corresponding subspace to see whether they contain the search point. The range search algorithm is a straightforward generalization.

BSP-trees can adapt well to different data distributions. However, they are typically not balanced and may have very deep subtrees, which has a nega-

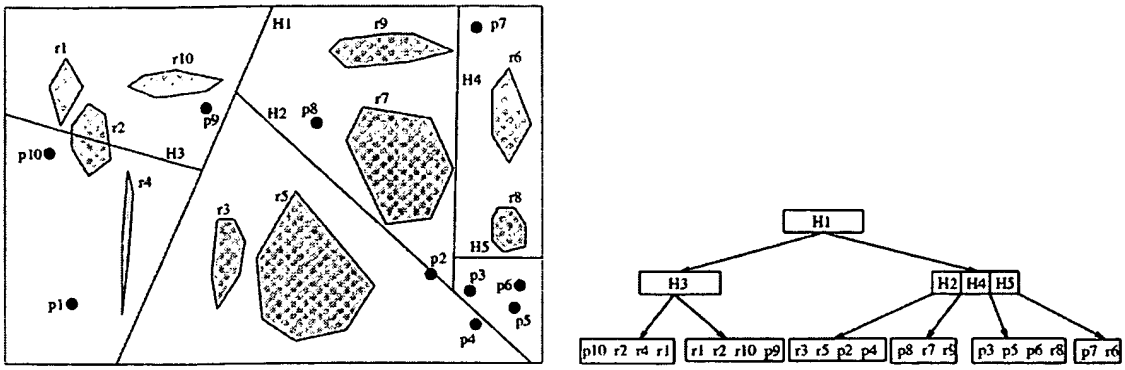


Figure 39. Cell tree.

inal object. The components do not have to be mutually disjoint. All components are assigned the same object identifier and inserted into the cell tree one by one. Due to clipping, we may have to subdivide each component into several cells during insertion, because it overlaps more than one subspace. Each cell is stored in one leaf node of the cell tree. If an insertion causes a disk page to overflow, we have to split the corresponding subspace and cell tree node and distribute its descendants between the two resulting nodes. Each split may propagate up the tree.

For point searches, we start at the root of the tree. Using the underlying BSP partitioning, we identify the subspace that includes the search point and continue the search in the corresponding subtree. This step is repeated recursively until we reach a leaf node, where we examine all cells to see whether they contain the search point. The solution consists of those objects that contain at least one of the cells that qualify. A similar algorithm exists for range searches. A performance evaluation of the cell tree [Günther and Bilmes 1991] shows that it is competitive with other popular spatial access methods.

Figure 39 shows our running example with five partitioning hyperplanes, each of them stored in the interior nodes. Even though the partitioning by means of the BSP-tree offers more flexibility than rectilinear hyperplanes, clipping objects may be inevitable. In Figure 39, we had to split r2 and insert the resulting cells into two pages.

As do all structures based on clipping, the cell tree has to cope with the fragmentation of space, which becomes increasingly problematic as more objects are inserted into the tree. After some time, most new objects will be split into fragments during insertion. To avoid the negative effects of this fragmentation, Günther and Noltemeier [1991] proposed the concept of *oversize shelves*. Oversize shelves are special disk pages attached to the interior nodes of the tree that accommodate objects which would have been split into too many fragments if they had been inserted regularly. The authors propose a dynamically adjusting threshold for choosing between placing a new object on an oversize shelf or inserting it regularly. Performance results of Günther and Gaede [1997] show substantial improvements compared to the cell tree without oversize shelves.

5.4 Multiple Layers

The multiple layer technique can be regarded as a variant of the overlapping regions approach, because data regions of different layers may overlap. However, there are several important differences. First, the layers are organized in a hierarchy. Second, each layer partitions the complete universe in a different way. Third, data regions within a